

TrailBlazer: creating and following paths along the web

Adam Mikeal

Department of Computer Science
Texas A&M University
adam@cs.tamu.edu

ABSTRACT

Since the concept was first proposed by Bush, the idea of *trails* or *paths* in hypertext has not received widespread public adoption. This is partially due to the lack of a popular, easy-to-use tool for the creation and traversal of paths, using familiar, common metaphors and providing an engaging interface. I propose to synthesize current research in hypertext trails into an dedicated client application—currently called TrailBlazer—that focuses on the human factors involved in the creation, publishing, and traversal of trails.

INTRODUCTION

Vannevar Bush initially posited the concept of a “trail” of documents as an ancillary to his Memex [1], and he described a system that provided two core features: the ability to easily store a train of thought through documents for later personal recall, and the ability to distribute (or publish) this trail to others. This concept is deceptively simple, yet offers so much in the way of data communication and recall. Interestingly, however, more than 50 years after Bush’s conceptual impetus there still does not exist a popular, commonly-available tool for this task. None of the major web browsers or word processor applications (the two primary influences on document and information workflow today) have ever, to my knowledge, attempted to integrate this concept into their core feature set.

So, although the process of following an idea along a “path” (and onto side-tracks, and dead ends, etc.) is an action performed by many people every day, because of the lack of integrated support for this process, users resort to a variety of coping mechanisms in order to encode these structures in a meaningful way. Web browser bookmarks, web applications made to replace

bookmark lists, many times simply emailing themselves or others a link or list of links to relevant sites. An educator desiring to provide her students with a set of related, relevant sites about the current subject matter might type up a list of URLs, and hand out a printed copy.

Although there has been significant research into the concepts and theories of document trails [3, 4, 5, 11, 12, 13], less attention has been paid to the human factors involved in interaction with the trail creation and traversal mechanisms. Since users are much less likely to adopt a new technology unless it is easier to implement than their current methods (or at the very least, requires no additional effort on their part) [10], it would suggest that implementation of a trails-based system should be approached from a human-centered perspective. I expect that such an implementation would be received by target users at the least with neutrality, if not encouragement.

Domain Focus

For this research proposal, I am choosing to limit the focus of the study to a specific user domain, for two reasons: 1) a narrower focus will provide tighter constraints on the feature set, limiting the project to a manageable size and scope, and 2) user testing over a smaller cross-section of users will provide better results with a smaller sample size.

Because of the many obvious applications, such as those discussed in [11, 12], I have decided to place the context of this project in the educational domain, providing both teachers and students with the ability to author, publish, and travel trails along the web in a direct, straightforward manner.

That stated, I believe that the improvements identified in this study will benefit user groups

outside an educational environment; the specifics, however, would be a topic for further research.

PROPOSAL

TrailBlazer will be a dedicated, native client whose sole purpose is the creation, publication, and traversal of trails. These trails will be an ordered, related group of web documents or sites that form a cohesive unit (usually based around a common theme). Interaction with the application will be centered around these tasks, not the general browsing task, and the interface will reflect this decision.

Technologies

I have chosen to implement TrailBlazer as a native application rather than a web-based solution, or a browser add-on for several reasons, primarily for the rich control set and greater amount of control over the user interaction you get with a native application. Since the focus of this project is to create a trails system that is both useful and *appealing* to users, these considerations are important. Secondly, because of the graphical nature of the interface needs, development time in a native application will likely be faster, hence more cost efficient.

Once the decision had been reached to implement a native application (as opposed to a web application or browser extension), the OS platform was the next choice. Mac OS X was chosen for four reasons: 1) the free development tools included with the operating system (Xcode IDE, gcc, Project Builder, etc.) make development low-cost and efficient, 2) the large number of bundled frameworks shipped with OS X eliminate a good deal of work, especially when implementing a browser-based application (Apple's Web Kit Framework can be implemented into a working browser in less than 10 lines of code), 3) OS X has an established set of human interface guidelines that help maintain consistency and usability across most Mac applications, and 4) the (relatively) high market penetration of Macs into the educational environment eased concerns about limiting access because of a platform decision.

Objective-C was chosen for the programming language because of its integration with, and ex-

cellent support for the Framework APIs used by the project.

A final benefit to choosing the Mac platform is its interesting ability to "package" a directory so that it appears to a user as a single file, which when clicked will simply launch an associated application. This opens up the possibility of storing alternate data along with the trail information, which we'll examine later.

2-section display

TrailBlazer's primary interface to the user will be a two-paned window consisting of a main browser window and a trail overview. This is implemented using the OS X control known as a "drawer", which can be retracted into the main window if desired, but will usually be visible, and sized to about half the width of the main browser window, so that the two components together create an entire window that is split into approximately $\frac{2}{3}$ main window, $\frac{1}{3}$ trail drawer (see Figure 1).

The overview of the trail shown in the trail drawer consists of a thumbnail representation of each node in the trail, loaded at the time that the user opens the trail. This provides an added benefit that the pages (each node in the trail) will all load quickly when visited. Another feature of the trail overview is the variable size of the nodes, with the largest node being the node that currently has focus in the main browser window (the current "stop" along the trail); refer again to Figure 1 for an example. The nodes that precede the follow the current node get successively smaller in size, as if they were receding on the horizon. This helps reinforce the metaphor of the trail, while simultaneously providing a layered effect with the information in the trail overview; the most important information (where you are now, where you just were, and where you're about to be) is the easiest to see, with the other nodes becoming gradually smaller into the distance.

This effect works just as well with branching paths as it does for simple linear paths; Figure 2 shows a possible visualization of a multiple-branching path; in fact, it would be possible to permit infinite branching, and simply render a



Figure 1. A prototypical TrailBlazer window in navigate mode, with a simple linear trail and the current node showing annotations.

few branches at once, with a control to “rotate” the direction of the visualization, but this would be problematic for two reasons: 1) the visualization would become quite complex very quickly, and thus defeat one of the applications primary goals: simplicity. 2) Even if the cognitive overhead of the visualization were overcome by the user, the “lost in hypertext” problem would just as quickly arise as the user navigated what was surely to be a complex graph structure. For these reasons it would probably be necessary to limit the branching factor of a path to an arbitrary number, to be determined in the user testing phase of the project evaluation.

Usage modes

In order to facilitate ease of both navigation along pre-existing paths, and path creation, the application will operate in two distinct modes, *Navigate*, for following along another’s path, and *Blaze*, for blazing new paths through the wilderness of the web.

Navigate. In Navigate mode, the primary focus of the user is traveling through the path; therefore, certain operations that are unnecessary to accomplish this goal are removed. For example, there is no visible address bar or URL indicator—primary navigation is accomplished through the trail drawer to the right or left of the main browser window. Other common features of a web browser, such as the “Home” button and a quick



Figure 2. An example of a trail overview drawer with a multiple-branching trail.

links toolbar, have also been eliminated. Initially, it was planned that the “Back” and “Forward” buttons, along with the “Stop” and “Reload” buttons, would also be removed, but discussions with colleagues have raised concerns about creating confusion among users with a lack of interface consistency, since TrailBlazer will undoubtedly be identified by most users as a type of web browser, and almost all modern web browsers implement these functions as buttons. This issue should be resolved with the formative evaluation.

The single toolbar provided to the user while in Navigate mode will contain two buttons to allow forward and backwards navigation through the trail (irrespective of previous actions). Potentially the standard set of four basic “browser functions” will also be included: stop, reload, back, and forward. The UI difficulty here will be clearly distinguishing to the user the difference between the trail previous and next buttons, and the browser forwards and back buttons. While the browser buttons will always take you to the most recent item in the browser’s cache, the trail buttons will take you to the next ordered node along the trail, even if you have never seen it before. For example, if a user jumps from node 1 to node 3, the browser back button would take them back to node 1, but the trail previous button would take them to node 2, which is the predecessor to node 3.

Blaze. To move from Navigate to Blaze mode, the user would select “Blaze Mode” from the Edit menu and would then be presented with a dialog box containing a binary option: “Edit an existing path or Blaze a New Trail?” If the user selects to edit an existing path, a file selection dialog appears, and that trail is loaded for editing. Otherwise, TrailBlazer is reset, clearing the trail drawer and exposing the editing toolbar, which adds a URL box, a search box, and a “Show Inspector” button to the default set of tools.

Creating and editing trails is remarkably similar to browsing them; the main difference being that the trail overview drawer is now “unlocked”, and the thumbnails representing each node can be dragged around at will to re-arrange the order, dragged off the drawer completely to remove the node from the trail in a puff of smoke, or a new node added into the trail by dragging from the main browser window into the trail drawer. All the basic operations of trail creation, maintenance, and ordering are straightforward and intuitive.

Blazing new trails, con’t.

Of course, there is more to creating a trail than simply placing a set of nodes in a sequence. Trails can have scripted actions, can be set up to run automatically, and each node could have properties and conditions that are independent of the

others [13]. In TrailBlazer, these properties are all set using the Inspector.

Maintaining consistency with other OS X applications like Keynote and Pages, the Inspector window floats outside the main TrailBlazer window and is independent of it. Only available during Blaze mode, the inspector allows a trail author to specify attributes for the currently highlighted node in the trail, and set actions, scripting events, conditional statements (the navigator must stay on this node for n seconds before being allowed to travel away), and also set properties for the trail as a whole, such as the default stop length if the trail is played back automatically, whether the annotations are visible, whether the navigator can make their own annotations, and whether the browsing is locked to the trail (disallowing off-path browsing).

Another task the inspector could perform would be monitoring the change rate of the content across various axes, such as content changes, structural changes, presentation changes, etc. [4]. This could be presented to the author/editor with a visual or graphic display, and perhaps a color highlighting around each node in the trail drawer to indicate its change rate since the last time the author worked on the trail. The manner in which this is accomplished is related directly to the publishing mechanism.

Publication

The OS X package model was mentioned earlier as an example of the benefits of developing on the Mac platform. It is in the publishing stage of TrailBlazer that this feature is used to full effect. Because the package file is really just a directory that has been granted special status by the OS X Finder, anything can be put inside; indeed, most Cocoa applications are packaged this way in a .App file.

TrailBlazer will create a new package for each trail that it saves, such as MyPath.trail (.trail in this case being the extension for our customized package). Inside this package (directory) will be a directory for every node in the trail, and those directories will contain a cached copy of the site, including images, HTML, style-sheets, and any

other assorted media needed to reproduce the page that was included in the trail. There is no cost to gather this, as it was already collected when the trail drawer rendered the thumbnail. This cached copy is saved in the trail along with a timestamp to indicate when the cached copy was collected. The next time a user opens the trail (for navigation or editing), TrailBlazer will attempt to get a newer version of the page (unless the author has instructed it not to, of course). If there is no newer version available, it does nothing. If the site cannot be found (an HTTP 404 error), that node is flagged and the editor is notified the next time that trail is opened in Blaze mode.

The result is that once the trail is saved and published to others, the nodes will never grow stale, no matter what actually happens to the original site. Also, network connectivity is only required during the creation stage; the trail can be easily traversed offline, since there is a cached copy of every node. This opens up many possibilities for trail publication to rural and disadvantaged schools where network connectivity might be sporadic, at best. A .trail file could simply be burned onto a CD or dropped on a USB drive and read anywhere.

Versioning

Along with the benefits of offline viewing provided by TrailBlazer's caching technique, the ability to keep versions of each iteration of the trail would be almost automatic. Each time the trail is opened, if it detects that there is a newer version of a node, it can simply save that new version to a different location within the .trail package. By associating each new cache version with a date, and trail could easily be opened and navigated as the trail that it was 6 months ago, or last year. Of course, the versioning could be turned on or off, based on the preferences of the particular trail's author.

Annotations

We are provided one more "freebie" from the OS X package file format; since we are able to include any amount of additional information with the trail, we can fairly easily include annotations on each node from the trail's author. These annotations can be stored as Rich Text files, to allow for

arbitrary markup, and dropped in the directory along with the cached copy of the site which the node represents. When the node is rendered into the main browser window, if there are any annotations available, they will be rendered in an adjustable frame at the bottom of the window (see Figure 1).

Additionally, with very little effort TrailBlazer could also support user annotations to any node in the trail; with a click of a button on the annotation reader's frame it converts to a rich text editor, complete with a simple toolbar for basic text formatting (while visually not changing much at all). The user could then create notes as they wished, perhaps responding to questions posed by the trail author (usually a teacher), and saving the annotations back to ultimately create a new version of the trail. TrailBlazer would save these annotations to the user's home directory by default, and also provide an export functionality so the user could publish the annotated trail as a new, unique document.

Developing the trail metaphor

There are many possibilities within this application concept to further develop the "trail" metaphor that I have started. For trails that are shared among peers, a mechanism to show which nodes receive the most amount of attention (either number of times viewed or number of visits received) could be useful to see what information others are finding interesting. This could be represented as increased "wear" along the path, just as well-worn paths give a visual indicator of frequency of use [2]. Multi-branching paths could be rendered in such a way that the forks in the path, the decision-points, gave the impression of actual forks in a road or path through the woods.

Suffice to say that the metaphor developed by TrailBlazer is rich, and certainly not fully explored in this research project.

PRIOR WORK

TrailBlazer is hardly the first application that supports the creation of trails. Indeed, it offers very little in the way of innovative research in the field of hypertext trails, if anything; it's distinctive feature is its approach toward the issue—a human

centered approach, that emphasizes the usability of the interface.

Early implementations of document trails were found in closed hypermedia systems, such as the HyperTalk scripts used to create paths through a document set in HyperCard. NoteCards also had a mechanism for demarcating a path, called *guided tours*, that initially had the purpose of communicating the organization of a filebox to users unfamiliar with its contents [13].

Zellweger's Scripted Documents provides a complete hypermedia system in which *all* links are represented as paths among documents. She emphasized the scriptability of the nodes, and the concept of automatic playback of the path. In many ways her writing provided the inspiration for TrailBlazer; many of the difficulties I've attempted to address in it's design were raised by Zellweger in her section on future work, including better support for visualization and navigation, tools for creators as well as readers, and representation of multiple branches along a single path [13].

The basic concept of linking various scattered web sites together to form an easily-navigable, cohesive trail, as well as the addition of annotation to the nodes belongs to Walden's Paths [5, 12]. While lacking some of the visualization techniques that distinguish TrailBlazer, most of the basic traversal functionality is there; and there is a wealth of research into its use in the classroom environment, along with many pre-constructed trails on many topics, from geologic formations to the plight of the American wolf.

TrailBlazer is a hypertext/hypermedia organizer, but it is primarily a graphic tool, not a textual one. The ideas for the unique visualization came from many sources, but in particular I should mention the OmniWeb web browser from The Omni Group [8]. The initial concept of using thumbnail representations of the various nodes was inspired by their novel implementation of tabbed browsing using thumbnails; their browser also served as an adequate model for the initial prototype mock-ups.

EVALUATION

In keeping with the concept of early focus on users and integrated design, I propose to evaluate TrailBlazer in three stages: initially, before the first prototype is built with a heuristic evaluation; then with user testing during the first design iteration to serve as a formative evaluation, and allow for incorporation of the results into the design process immediately; finally, after the product is finalized, a formal design evaluation will take place, using a task-based user tests.

While the final product evaluation is ostensibly the metric used to determine if the concept was successful in its stated goals, by focusing on usability evaluations earlier in the design process, the likelihood of reaching that success should be greatly increased [6].

Heuristic evaluation

The initial evaluation of the system will be sans users, with a detailed set of heuristics. Much thought has gone into the initial design of TrailBlazer already, and likely there will be little change from this stage of the evaluation, but it has been shown that a heuristic analysis is one of the most cost-effective usability studies available, with a very high benefit-cost ratio [9]. It is likely that small changes in design philosophy or interface layout at this stage in the implementation would save vastly greater amounts of time later on in the process. Even applying a simple set of heuristics like Nielsen's 10 Usability Heuristics would highlight inconsistencies in the interface, potential stumbling blocks for users, and incorrect assumptions [8].

There are already issues that have been identified as questions that will require actual user studies to resolve, such as the organization and placement (or not) of the two different types of navigation buttons on the main browser window: the browser forward and back, as well as the trail previous and next buttons. There will likely be others that arise during the design process.

Formative evaluation

Rather than wait to do formal user testing after the product is built, an iterative design philosophy suggests that running a series of formative

evaluations *during* the design process could help to shape the course of the product design in a user-centered way. I estimate that there will be several (four to six) iterations of design and user testing, each iteration incorporating the latest results from the usability tests.

The end result of this cycle will be a product design that has a higher level of usability than otherwise probable, along with a continual re-focusing of the design philosophy back to the user.

Formal design evaluation

The final stage of the product evaluation for TrailBlazer will be the task-based user tests, where users will be given a directed set of tasks, such as:

- Collect a set of documents/websites about your favorite [artist|author|thinker] and compile it as if you were to pass the information on to a friend.
- Your boss has asked for some information on a new technology that will be impacting your employer. She wants you to prepare an informal presentation (no Powerpoint) for your group, and it needs to be something that they can take with them for more detailed reading later.
- When handed a list of links, how do you go about browsing the sites as a whole? how do you see them relating to each other? how do you store them for later recall?

Since this project has a particular emphasis on the use of TrailBlazer in educational environments, an attempt will be made specifically to recruit teachers for this stage of the evaluation. Their questions will be more targeted, for example:

- Collect a set of websites that deal with the current historical event you are studying in your classroom. Compile it into a unit, and annotate each site so that your students will have an understanding of where each site fits in the context of the others.

The users will then be timed during their execution of these tasks, first using whatever tools they would normally use to accomplish such tasks., such as browsers, text editors, word processors, email clients, databases, etc. Next, after a brief introductory period in TrailBlazer with minimal training, they will be asked to perform the same set of tasks using TrailBlazer (where appropriate, topics will be changed to eliminate any time advantage that might be gained in performing an identical task a second time). Finally, the users will be administered surveys both before and after the study to ascertain their attitudes toward using various technologies to accomplish the types of tasks represented in the study.

My expectation is that not only will the timed tests show an objective improvement in the time it takes the users to accomplish the particular tasks, but that there will a positive attitude shift toward using TrailBlazer for various tasks that currently don't map well to existing toolsets.

CONCLUSION

The concept of document trails has great potential, and there is a volume of excellent research literature on the subject, along with several reference implementations. What is lacking for widespread adoption by the public is an implementation that chooses to focus foremost on the user, and a design that integrates creation, publishing, and navigation of trails into one elegant interface.

Similar to Miles' concept of link excess [6], by imposing links between documents where there were none before, the trail author is recontextualizing the content found in each "stop", or node, along the trail in reference to the later documents encountered. In effect, she is constructing an extended argument or thought process spanning many individual lexia. While any task can stand to gain from an interface that incorporates well-thought-out design and usability principles, it is particularly true for tasks such as this, where high cognitive overhead can interfere with the creative task. Using an interface to create the trail that makes the creative act effortless and intuitive would go a long way toward encouraging users to adopt the technology, and promote the discovery and publication of new trails and paths

through our ever-expanding, increasingly complex mediascape.

REFERENCES

1. Bush, V. As We May Think. *The Atlantic Monthly*, July 1945, 101-108.
2. Chu, Y., Bainbridge, D., Jones, M., and Witten, I. H. Realistic Books: A Bizarre Homage to an Obsolete Medium? In *Proceedings of the Joint Conference on Digital Libraries 2004*, (Tuscon, AZ, July 2004), ACM Press, 78-86.
3. Dave, P., Karadkar, U. P., Furuta, R., Francisco-Revilla, L., Shipman, F., Dash, S., and Dalal, Z. Browsing Intricately Interconnected Paths. In *Proceedings of Hypertext 2003* (Nottingham U.K., August 2003), ACM Press, 95-103.
4. Francisco-Revilla, L., Shipman III, F. M., Furuta, R., Karadkar, U., and Arora, A. Perception of Content, Structure, and Presentation Changes in Web-based Hypertext. In *Proceedings of Hypertext 2001*, (Aarhus, Denmark, August, 2001), ACM Press, 205-214.
5. Furuta, Richard, Shipman III, Frank M., Marshall, Catherine C., Brenner, Donald, and Hsieh, H. Hypertext Paths and the World-Wide Web: Experiences with Walden's Paths. In *Proceedings of Hypertext 1997* (Southampton, U.K., April 1997), ACM Press, 167-176.
6. Gould, J. How to design usable systems. 1988, revised. From *Readings in Human-Computer Interaction: Towards the Year 2000*.
7. Miles, A. Hypertext Structure as the Event of Connection. In *Proceedings of Hypertext 2001*, (Aarhus, Denmark, August, 2001), ACM Press, 61-68
8. Nielsen, J. Ten Usability Heuristics. 2005, http://www.useit.com/papers/heuristic_list.html.
9. Nielsen, J. Guerrilla HCI: Using Discount Usability Engineering to Penetrate the Intimidation Barrier. 1994, .

http://www.useit.com/papers/guerrilla_hci.html.

10. Nielsen, J., and Norman, D. A. Usability On The Web Isn't A Luxury. In *InformationWeek*, (January 14, 2000).
11. Shipman, F., Furuta, R., Brenner, D., Chung, C., and Hsieh, H. Guided Paths through Web-Based Collections: Design, Experiences, and Adaptations. In *Journal of the American Society of Information Sciences (JASIS)* 51(3) (March 2000), 260-272.
12. Shipman III, F. M., Marshall, C. C., Furuta, R., Brenner, D. A., Hsieh, H and Kumar, V. Creating Educational Guided Paths over the World-Wide Web. In *Educational Telecommunications, 1996: Proceedings of ED-TELECOM 96*, (June 1996), 326-331.
13. Zellweger, P. T. Scripted Documents: A Hypermedia Path Mechanism. In *Proceedings of Hypertext 1989* (New York NY, November 1989), ACM Press, 1-14.
14. The Omni Group. Seattle, WA.
<http://www.omnigroup.com/>.